Computational Intelligence, Volume 59, Number 000, 2010

Undirected Dependency Parsing

CARLOS GÓMEZ-RODRÍGUEZ
cgomezr@udc.es
Depto. de Computación, Universidade da Coruña. Facultad de Informática, Campus de Elviña s/n, 15071 A Coruña, Spain

DANIEL FERNÁNDEZ-GONZÁLEZ
danifg@uvigo.es
Depto. de Informática, Universidade de Vigo. Campus As Lagoas 32004 Ourense, Spain

VÍCTOR MANUEL DARRIBA BILBAO
darriba@uvigo.es
Depto. de Informática, Universidade de Vigo. Campus As Lagoas 32004 Ourense, Spain

Dependency parsers, which are widely used in natural language processing tasks, employ a representation of syntax in which the structure of sentences is expressed in the form of directed links (dependencies) between their words. In this article, we introduce a new approach to transition-based dependency parsing in which the parsing algorithm does not directly construct dependencies, but rather undirected links, which are then assigned a direction in a post-processing step. We show that this alleviates error propagation, since undirected parsers do not need to observe the single-head constraint, resulting in better accuracy.

Undirected parsers can be obtained by transforming existing directed transition-based parsers as long as they satisfy certain conditions. We apply this approach to obtain undirected variants of three different parsers (the Planar, 2-Planar and Covington algorithms) and perform experiments on several datasets from the CoNLL-X shared tasks, showing that our approach is successful in reducing error propagation and produces improvements in parsing accuracy in most of the cases, achieving results competitive with state-of-the-art transition-based parsers.

Key words: natural language processing, dependency parsing, parsing, computational linguistics, automata

This is the **pre-peer reviewed version** of the following article: Carlos Gómez-Rodríguez, Daniel Fernández-González and Víctor M. Darriba, *Undirected Dependency Parsing*, Computational Intelligence, 31(2):348-384, 2015 (ISSN 1467-8640); which has been published in final form at:
http://onlinelibrary.wiley.com/doi/10.1111/coin.12027/abstract

1. INTRODUCTION

Syntactic parsing is the process of determining the grammatical structure of a sentence: given an input sentence, a parsing algorithm (or parser) will analyze it in order to output a representation of its underlying structure. The format of this representation and the information it contains depends on the particular syntactic theory used by the parser. In constituency parsers, or phrase structure parsers, sentences are analyzed by breaking them down into meaningful parts called constituents, which are in turn divided into smaller constituents. The result of such an analysis is represented with a constituency tree, like the one shown in Figure 1. On the other hand, in dependency parsers, the structure of the sentence is represented by a set of directed links (called dependencies) between its words, forming a graph like the one in Figure 2.

Dependency parsing has gained wide popularity in the natural language processing com-

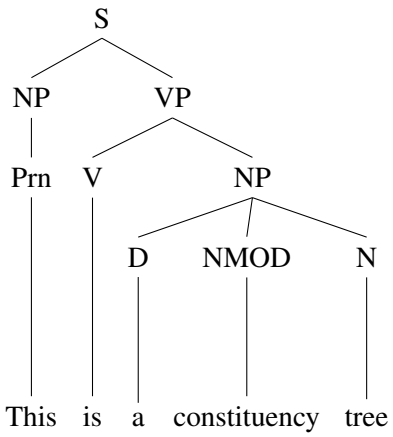

FIGURE 1. Constituency tree for an English sentence.

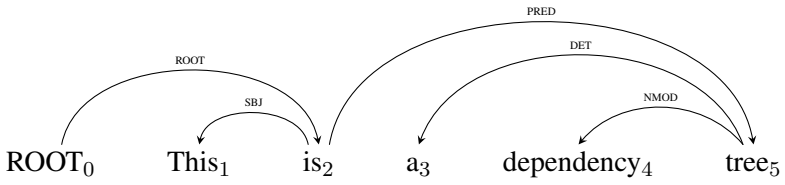

FIGURE 2. Dependency tree for an English sentence.

munity, and it has recently been applied to a wide range of problems, like machine translation (Ding and Palmer, 2005; Shen et al., 2008; Xu et al., 2009; Katz-Brown et al., 2011), textual entailment recognition (Herrera et al., 2005; Berant et al., 2010), relation extraction (Culotta and Sorensen, 2004; Fundel et al., 2006; Miyao et al., 2009; Katrenko et al., 2010), question answering (Cui et al., 2005; Comas et al., 2010), opinion mining (Joshi and Penstein-Rosé, 2009) or learning for game AI agents (Branavan et al., 2012).

Some important advantages of dependency representations over constituency trees when applied to natural language processing tasks are that they provide a more explicit representation of the semantic information that is useful for applications (for example, the subject and object of a sentence in Figure 2 are explicitly represented in its dependency graph); they do not need intermediate nodes (non-terminals) and hence allow for simpler and more efficient parsing algorithms; and they represent discontinuous linguistic phenomena caused by long-range dependencies or free word order in a natural way by using crossing dependencies.

While there has been research in grammar-driven dependency parsing, where formal grammatical rules are used to define the set of dependency structures that can appear in a language (Tapanainen and Järvinen, 1997; Lombardo and Lesmo, 1996); most current dependency parsers are data-driven, i.e., they use learning techniques to automatically infer linguistic knowledge from annotated corpora, which can then be used to parse new sentences without the need for an explicit grammar.

In particular, the vast majority of data-driven dependency parsers that have been defined in recent years can be described as being either graph-based or transition-based dependency parsers (Zhang and Clark, 2008; McDonald and Nivre, 2011). Graph-based parsing uses global optimization on models that score dependency graphs (Eisner, 1996; McDonald et al., 2005). In transition-based parsing, which is the focus of this article, dependency graphs are

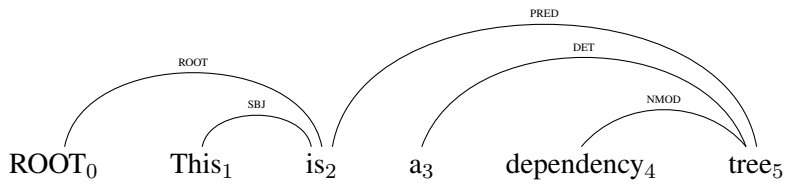

FIGURE 3. Undirected dependency graph for the sentence in Figure 2.

built by sequences of actions by an automaton that transitions between parser states, and each action is scored individually. These scores are used to find a suitable sequence of actions for each given sentence, typically by greedy deterministic search or beam search (Yamada and Matsumoto, 2003; Nivre et al., 2004a; Nivre, 2008). Some key advantages of transition-based parsers are their high efficiency (many of them have linear time complexity, while still providing state-of-the-art accuracy) and the possibility of easily incorporating arbitrarily rich feature models, including non-local features (Zhang and Nivre, 2011).

However, it has been shown by McDonald and Nivre (2007) that transition-based parsers suffer from error propagation: as the generation of a dependency parse for a sentence is modelled as a sequential process, an early erroneous decision may place the parser into an incorrect state, causing more errors later on. In particular, one source of error propagation in transition-based parsers is the need to enforce the single-head constraint, i.e., the common restriction in dependency syntax that forbids each node from having more than one incoming arc. For instance, if we are parsing the sentence in Figure 2 with a transition-based parser that uses greedy deterministic search and we mistakenly make a decision to build an arc from a_3 to tree_5 instead of the correct dependency from tree_5 to a_3; we will not only miss this dependency but also the one from is_2 to tree_5, since we will be in a parser state where the single-head constraint makes it illegal to create it (due to not allowing the node tree_5 to have two incoming arcs).

In this article, we introduce a new approach to transition-based parsing that improves accuracy by alleviating this kind of error propagation. To do so, we build novel *undirected* dependency parsers by modifying existing transition-based dependency parsers – namely, the Planar and 2-Planar parsers by Gómez-Rodríguez and Nivre (2013) and the non-projective list-based parser by Nivre (2008), which is a variant of the algorithm by Covington (2001).

The obtained undirected parsers are algorithms that build an undirected graph (like the one on Figure 3) rather than a dependency graph. This means that the single-head constraint need not be observed during the parsing process, since the directed notions of head and dependent (or of incoming and outgoing arcs) are lost in undirected graphs. Therefore, this gives the parser more freedom, and can prevent situations where enforcing the constraint leads to error propagation, like the previous example.

On the other hand, these new algorithms have the obvious disadvantage that their output is an undirected graph, and not a dependency graph. We will need a post-processing step to recover the direction of the dependencies, generating a valid dependency structure. Thus, some complexity is moved from the transition-based parsing process to this post-processing step; and each undirected parser will outperform the original directed version only if the simplification of the parsing phase is able to avoid more errors than are generated in the post-processing step. Fortunately, as we will see in the experimental sections, this is in fact the case for most of the algorithm-dataset combinations that we tried, showing that the undirected parsing approach is useful to improve parsing accuracy.

The remainder of this article is structured as follows: Section 2 introduces common notation and concepts regarding transition-based dependency parsing, which will be used

throughout the article. Section 3 describes a technique to transform transition-based dependency parsers satisfying certain conditions into undirected parsers, and applies it to the Planar, 2-Planar and Covington algorithms. In Section 4, we discuss two different post-processing techniques that can be used to recover dependency trees from undirected graphs. Section 5 puts the ideas in previous sections into practice and presents an empirical study of the accuracy of undirected dependency parsers compared to their directed counterparts, and Section 6 presents error analysis to see the effect of undirected parsing on error propagation. Finally, Section 7 concludes the article with a discussion of the results.

2. TRANSITION-BASED DEPENDENCY PARSING

We now introduce some definitions and notation concerning transition-based dependency parsing, which will serve as a basis to present our undirected parsing approach.

2.1. Dependency parsing and dependency graphs

A *dependency parser* is a system that analyzes natural language sentences and outputs a representation of their syntactic structure in the form of a dependency graph, like the one shown in Figure 2. More formally, a dependency graph can be defined as follows:

Definition 1: Let $w = w_1 \dots w_n$ be an input sentence. Given a set L of labels, a *dependency graph* for $w_1 \dots w_n$ is a labelled directed graph $G = (V_w, A)$, where $V_w = \{0, \dots, n\}$ is the set of nodes, and $A \subseteq V_w \times L \times V_w$ is the set of labelled directed arcs.

Each of the nodes in V_w encodes the position of a token in w, except for the node 0 which is a dummy node used to mark the root of the sentence and cannot have incoming arcs. Each of the arcs in A encodes a dependency relation between two tokens. We will write the arc $(i, l, j) \in A$ as $i \xrightarrow{l} j$, which will also be called a *dependency link* labelled l from i to j. We then say that i is the *head* of j and, conversely, that j is a syntactic *dependent* of i. The labels on dependency links are typically used to represent their associated syntactic functions, such as SBJ for subject in Figure 2.

Given a dependency graph $G = (V_w, A)$, we will write $i \rightarrow j \in A$ if there is a dependency link from i to j, regardless of its label. We will write $i \rightarrow^\star j \in A$ if there is a (possibly empty) path from i to j; and $i \leftrightarrow^\star j \in A$ if there is a (possibly empty) path connecting i and j in the undirected graph underlying G. When using these notational conventions, we will omit the references to A when the relevant set of edges is clear from the context.

Most dependency-based syntactic formalisms do not allow arbitrary dependency graphs as syntactic representations. Instead, they are typically restricted to acyclic graphs where each node has at most one head. Such dependency graphs are called *dependency forests*.

Definition 2: A dependency graph G is said to be a *dependency forest* if it satisfies the following:

(1) *Single-head constraint*: if $i \rightarrow j$, then there is no $k \neq i$ such that $k \rightarrow j$.
(2) *Acyclicity constraint*: if $i \rightarrow^\star j$, then there is no arc $j \rightarrow i$.

Nodes that have no head in a dependency forest are called *roots*. Apart from the previous two constraints, some dependency formalisms add the additional constraint that a dependency forest can have only one root (or, equivalently, that it must be connected). A forest of this form is called a *dependency tree*.

In this article, we will work with dependency parsers that output dependency forests, i.e., that enforce the single-head and acyclicity constraints. While we do not require the parsers to explicitly enforce connectedness, we note that any dependency forest can be easily converted to a dependency tree (and thus made connected) by linking the dummy root node 0 as the parent of all its other root nodes. Therefore, from now on, we will refer to dependency trees and forests interchangeably.

2.2. Transition systems

In a transition-based dependency parser, the dependency analysis for an input sentence is built by a non-deterministic state machine that reads the input sentence and builds dependency arcs. Following the framework of Nivre (2008), this non-deterministic automaton is called a *transition system*, and is defined as follows:

Definition 3: A *transition system* for dependency parsing is a tuple $S = (C, T, c_s, C_t)$, where

(1) C is a set of possible parser *configurations*,
(2) T is a finite set of *transitions*, which are partial functions $t : C \rightarrow C$,
(3) c_s is a total initialization function that maps each input string w to a unique *initial configuration* $c_s(w)$, and
(4) $C_t \subseteq C$ is a set of *terminal configurations*.

Although the specific nature of configurations varies among parsers, they are required to contain at least a set A of dependency arcs and a buffer B of unread words, which initially holds all the words in the input sentence. A transition-based parser will be able to read input words by popping them from the buffer, and to create dependency arcs by adding them to the set A.

Given an input string $w = w_1 \dots w_n$, each of the sequences of configurations that the transition system S can traverse by sequentially applying transitions starting from the initial configuration $c_s(w)$ and ending at some terminal configuration $c_t \in C_t$ is called a *transition sequence* for w in S. The *parse* assigned to w by a transition sequence ending at the terminal configuration c_t is the dependency graph $G = (\{0, \dots, n\}, A_{c_t})$, where A_{c_t} is the set of arcs stored in the configuration c_t.

Note that, from a theoretical standpoint, it is possible to define a transition system such that no transition sequences exist for a given sentence w. However, this is usually avoided in practice, since robustness (the ability to terminate producing a parse for every possible input) is seen as a desirable quality in data-driven natural language parsers. Therefore, all the transition systems that we will use and define in this article have the property that there is at least one (and typically more than one) transition sequence for every sentence w.

In order to use a transition system to obtain the best dependency analysis for a given sentence, we need to have a mechanism that will select the most suitable among all the transition sequences that the system allows for that sentence. A standard method to achieve this is by using a classifier to select the best transition to execute at each configuration.

To do so, we define an *oracle* for the transition system $S = (C, T, c_s, C_t)$ as a function $o : C \rightarrow T$, i.e., an oracle is a function that selects a single transition to take at each configuration, and thus can be used to determinize the parsing process. Given a training treebank containing manually annotated dependency trees for sentences, we train a classifier to approximate an oracle by building a canonical transition sequence for each tree in the treebank, and using each of the configurations in the sequence and the corresponding chosen transition as a training instance.

Then, to parse a sentence w, we only need to initialize the parser to the initial configuration $c_s(w)$ and iteratively apply the transitions suggested by the classifier until a terminal configuration is reached. This results in a parser that performs a greedy deterministic search for the best transition sequence, one of the most widely used approaches in transition-based parsing (Yamada and Matsumoto, 2003; Nivre et al., 2004b; Attardi, 2006; Nivre, 2008; Gómez-Rodríguez and Nivre, 2010; Goldberg and Elhadad, 2010; Tratz and Hovy, 2011; Gómez-Rodríguez and Nivre, 2013); although other optimization and search strategies, such as beam search, can also be used (Johansson and Nugues, 2006; Titov and Henderson, 2007; Zhang and Clark, 2008; Huang et al., 2009; Huang and Sagae, 2010; Zhang and Nivre, 2011; Hayashi et al., 2012).

2.3. An example transition system: The Planar transition system

A simple example of a practical transition system for dependency parsing is the *Planar* transition system, introduced by Gómez-Rodríguez and Nivre (2010). The Planar parser is an extension of the well-known arc-eager projective parser by Nivre (2003), which can handle all dependency trees that are *planar*, i.e., those whose arcs can be drawn above the words (as in Figure 2) in such a way that no two arcs cross. In contrast, the arc-eager parser by Nivre (2003) can only build so-called *projective* trees, which is a slightly more restricted set of syntactic structures (Gómez-Rodríguez and Nivre, 2010).

The Planar transition system is a transition system $S = (C, T, c_s, C_t)$ such that

- C is the set of all configurations of the form $c = \langle \sigma, B, A \rangle$, where σ and B are disjoint lists of nodes from V_w (for some input w), and A is a set of dependency arcs over V_w. The list B, called the *buffer*, is used to hold nodes corresponding to input words that have not yet been read. The list σ, called the *stack*, contains nodes for words that have already been read, but still have dependency links pending to be created. For perspicuity, we will use the notation $\sigma|i$ to denote a stack with top i and tail σ, and the notation $j|B$ to denote a buffer with top j and tail B. The set A of dependency arcs contains the part of the output parse that the system has constructed at each given point.
- The initial configuration is $c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, i.e., the buffer initially holds the whole input string while the stack is empty.
- The set of terminal configurations is $C_t = \{\langle \sigma, [], A \rangle \in C\}$, i.e., final configurations are those where the buffer is empty, regardless of the contents of the stack.
- The set T has the following transitions:

SHIFT	$\langle \sigma, i\|B, A \rangle \Rightarrow \langle \sigma\|i, B, A \rangle$
REDUCE	$\langle \sigma\|i, B, A \rangle \Rightarrow \langle \sigma, B, A \rangle$
LEFT-ARC$_l$	$\langle \sigma\|i, j\|B, A \rangle \Rightarrow \langle \sigma\|i, j\|B, A \cup \{j \xrightarrow{l} i\} \rangle$ only if $\nexists k \mid k \to i \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
RIGHT-ARC$_l$	$\langle \sigma\|i, j\|B, A \rangle \Rightarrow \langle \sigma\|i, j\|B, A \cup \{i \xrightarrow{l} j\} \rangle$ only if $\nexists k \mid k \to j \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).

The SHIFT transition is used to read words from the input string, by moving the next node in the buffer to the top of the stack. The LEFT-ARC and RIGHT-ARC transitions build leftward and rightward dependency arcs, respectively, connecting the first node on the buffer and the topmost node on the stack. Finally, the REDUCE transition is used to pop the topmost node from the stack when we have finished building arcs to or from it.

Figure 4 shows a transition sequence in the Planar transition system which derives the labelled dependency graph in Figure 2.

Note that the Planar parser is a linear-time parser, since each word in the input can be shifted and reduced at most once, and the number of arcs that can be built by LEFT-ARC

Transition	**Stack** (σ)	**Buffer** (B)	**Added Arc**
	[ROOT$_0$]	[This$_1$,...,tree$_5$]	
SHIFT	[ROOT$_0$, This$_1$]	[is$_2$,..., tree$_5$]	
LA$_{SBJ}$	[ROOT$_0$, This$_1$]	[is$_2$,..., tree$_5$]	(2, SBJ, 1)
REDUCE	[ROOT$_0$]	[is$_2$,..., tree$_5$]	
RA$_{ROOT}$	[ROOT$_0$]	[is$_2$,..., tree$_5$]	(0, ROOT, 2)
SHIFT	[ROOT$_0$, is$_2$]	[a$_3$,..., tree$_5$]	
SHIFT	[ROOT$_0$, is$_2$, a$_3$]	[dependency$_4$, tree$_5$]	
SHIFT	[ROOT$_0$, is$_2$, a$_3$, dependency$_4$]	[tree$_5$]	
LA$_{NMOD}$	[ROOT$_0$, is$_2$, a$_3$, dependency$_4$]	[tree$_5$]	(5, NMOD, 4)
REDUCE	[ROOT$_0$, is$_2$, a$_3$]	[tree$_5$]	
LA$_{DET}$	[ROOT$_0$, is$_2$, a$_3$]	[tree$_5$]	(5, DET, 3)
REDUCE	[ROOT$_0$, is$_2$]	[tree$_5$]	
RA$_{PRED}$	[ROOT$_0$, is$_2$]	[tree$_5$]	(2, PRED, 5)
SHIFT	[ROOT$_0$, is$_2$, tree$_5$]	[ ]	
REDUCE	[ROOT$_0$, is$_2$]	[ ]	
REDUCE	[ROOT$_0$]	[ ]	

FIGURE 4. Transition sequence for parsing the sentence in Figure 2 using the Planar parser (LA=LEFT-ARC, RA=RIGHT-ARC).

and RIGHT-ARC transitions is strictly bounded by the number of words by the single-head constraint.

2.4. The 2-Planar and Covington transition systems

The undirected dependency parsers defined and tested in this article are based on the Planar transition system described above, the 2-Planar transition system by Gómez-Rodríguez and Nivre (2010) and the version of the Covington (2001) non-projective parser defined by Nivre (2008). We now outline the two latter parsers briefly, a more comprehensive description can be found in the above references.

The 2-Planar transition system is an extension of the Planar system that can recognize a larger set of dependency trees, called 2-planar dependency trees. A dependency tree is said to be 2-planar if it is possible to draw it assigning one out of two colors to each of its dependency arcs, in such a way that arcs sharing the same color do not cross. Gómez-Rodríguez and Nivre (2010) have shown that well over 99% of the dependency trees in natural language treebanks fall into this set, making this parser practical for languages that contain a significant proportion of crossing links, so that planar and projective parsers fall short in coverage.

To handle 2-planar structures, the 2-Planar transition system uses two stacks instead of one, with each stack corresponding to one of the colors that can be assigned to arcs. At each given configuration, one of the stacks is said to be *active* (meaning that we are building arcs of that color), while the other is *inactive*. Configurations are thus of the form $c = \langle \sigma_0, \sigma_1, B, A \rangle$, where σ_0 is the active stack and σ_1 the inactive stack. The initial configuration is $c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$ and the set of terminal configurations is $C_t = \{\langle \sigma_0, \sigma_1, [], A \rangle \in C\}$, analogously to the Planar transition system. The system has the following transitions:

SHIFT	$\langle \sigma_0, \sigma_1, i\|B, A\rangle \Rightarrow \langle \sigma_0\|i, \sigma_1\|i, B, A\rangle$
REDUCE	$\langle \sigma_0\|i, \sigma_1, B, A\rangle \Rightarrow \langle \sigma_0, \sigma_1, B, A\rangle$
LEFT-ARC$_l$	$\langle \sigma_0\|i, \sigma_1, j\|B, A\rangle \Rightarrow \langle \sigma_0\|i, \sigma_1, j\|B, A \cup \{j \xrightarrow{l} i\}\rangle$ only if $\nexists k \mid k \rightarrow i \in A$(single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
RIGHT-ARC$_l$	$\langle \sigma_0\|i, \sigma_1, j\|B, A\rangle \Rightarrow \langle \sigma_0\|i, \sigma_1, j\|B, A \cup \{i \xrightarrow{l} j\}\rangle$ only if $\nexists k \mid k \rightarrow j \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
SWITCH	$\langle \sigma_0, \sigma_1, B, A\rangle \Rightarrow \langle \sigma_1, \sigma_0, B, A\rangle$

The SHIFT transition reads words from the input string exactly as in the Planar transition system, but in this case their corresponding nodes are placed into both stacks. REDUCE, LEFT-ARC and RIGHT-ARC work similarly as in the Planar parser, but they only take into account the active stack, ignoring the inactive one. Finally, a SWITCH transition is added that makes the active stack inactive and vice versa, allowing us to alternate between the two possible arc colors. Despite this added functionality, the 2-Planar parser still runs in linear time.

On the other hand, the Covington algorithm is a transition system that runs in quadratic time, but it has the advantage of being able to parse every possible dependency tree, without restrictions like planarity or 2-planarity. The basic algorithm was first described by Covington (1990, 2001). Nivre (2008) implements a variant of this strategy as a transition system, which is the version we use here.

Configurations in this system are of the form $c = \langle \lambda_1, \lambda_2, B, A\rangle$, where λ_1 and λ_2 are *lists* containing nodes associated with partially processed words, and B is the *buffer* of unprocessed words. The idea of the algorithm is that, after reading each given word, we can do a right-to-left traversal of *all* the nodes for already-read words in the input and create links between them and the first node in the buffer. This traversal is implemented by moving nodes from λ_1 (untraversed nodes) to λ_2 (already traversed nodes). After reading each new input word, all the nodes in both lists are moved back to λ_1 for a new right-to-left traversal to start, hence the quadratic complexity.

The algorithm starts with an initial configuration $c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset\rangle$, and will terminate in final configurations of the set $C_f = \{\langle \lambda_1, \lambda_2, [], A\rangle \in C\}$. The system has the following transitions:

SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A\rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A\rangle$
NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A\rangle$
LEFT-ARC$_l$	$\langle \lambda_1\|i, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{j \xrightarrow{l} i\}\rangle$ only if $\nexists k \mid k \rightarrow i \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
RIGHT-ARC$_l$	$\langle \lambda_1\|i, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{i \xrightarrow{l} j\}\rangle$ only if $\nexists k \mid k \rightarrow j \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).

The SHIFT transition advances the parsing process by reading the first node in the buffer B and inserting it at the head of a list obtained by concatenating λ_1 and λ_2, thus starting a new right-to-left traversal to find candidate nodes to be linked to the one now heading the buffer. This traversal is implemented by the other three transitions: NO-ARC is used when there is no dependency relation between the first node in the buffer and the head of the list α_1, and it moves the head of the list α_1 to α_2 without creating any arcs; while LEFT-ARC and RIGHT-ARC create a leftward (rightward) arc connecting the first node in the buffer and the

head of the list α_1, and then move the head of α_1 to α_2. The traversal will end when a new SHIFT transition is executed, signifying that no more arcs will be created involving the first node in the buffer and the nodes in α_1.

3. TRANSFORMING DIRECTED PARSERS INTO UNDIRECTED PARSERS

As mentioned in Section 2.2, practical implementations of transition systems use greedy search or beam search to find the best transition sequence (and thus obtain a dependency tree) for each given input. Since these strategies build transition sequences and dependency arcs in a sequential way from the beginning of the sentence to the end, early parsing decisions may condition and restrict later decisions, causing error propagation. McDonald and Nivre (2007) present an empirical study whose results highlight this phenomenon, showing that a transition-based parser tends to be more accurate than a graph-based parser on arcs that are built early in the transition sequence, but less acurate on arcs built later on.

In particular, one possible source of error propagation is the single-head constraint described in Definition 2. In order to return a valid dependency tree, a transition system must obey this constraint during the whole parsing process. This means that a transition that creates a dependency arc is permissible only if its application does not violate the single-head constraint, i.e., if it does not result in assigning more than one head to the same node. For instance, Figure 4 shows a transition sequence for the Planar parser that correctly parses a sample sentence, assigning it the dependency tree in Figure 2. However, in an alternative scenario where the classifier made a mistake in the eighth transition choosing to apply RA_{NMOD} instead of the correct choice LA_{NMOD}, this would result into building a dependency link from $dependency_4$ to $tree_5$ instead of the correct link from $tree_5$ to $dependency_4$. In turn, this would lead to a situation where creating the (correct) link from is_2 to $tree_5$ would be forbidden by the single-head constraint, as node $tree_5$ would already have an incoming arc. Therefore, in this example, a single erroneous choice of transition initially affecting a single dependency arc propagates to other arcs, due to the single-head constraint, causing at least two attachment errors in the output tree.

In order to remove this source of error propagation, we transform the Planar, 2-Planar and Covington transition systems into variants that build *undirected* graphs instead of directed dependency trees. The goal of this transformation is to allow transition-based parsers to work without needing to obey the single-head constraint. This will make these parsers less sensitive to error propagation, since they will be able to create arcs freely at any point in the parsing sequence, regardless of the existing arcs that have been created before.

The mentioned transformation consists in redesigning the transition systems so that they create dependency links without a direction. In this way, it is not necessary to observe the single-head contraint during the parsing process, since the directed concepts of head and dependent do not apply to undirected links. As a result, the output of these new variants is an undirected graph instead of a tree. This transformation has previously been described in Gómez-Rodríguez and Fernández-González (2012).

3.1. The Undirected Planar, 2-Planar and Covington transition systems

With the goal of obtaining undirected transition systems from the directed ones described in Sections 2.3 and 2.4, we replace the LEFT-ARC and RIGHT-ARC transitions in those systems (which create directed arcs in each direction) with a new transition (ARC) that builds an undirected link. This can be done because, in these three transition systems, the effect of the two directed transitions is the same except for the direction of the created link, so that their behaviour can be collapsed into one common transition: the undirected ARC transition.

In addition to this, the configurations of the undirected transition systems must be changed

so that the arc set A is a set of undirected edges, instead of directed arcs. Analogously to our notation for directed arcs, we will use the notation $i \overset{l}{-} j$ as shorthand for an undirected edge labelled l connecting the nodes i and j.

Furthermore, since the direction of the arcs is lost in undirected graphs, the preconditions of transitions that guarantee the single-head constraint are simply removed from the systems.

If we apply these transformations and leave the Planar, 2-Planar and Covington transition systems otherwise unchanged, we will obtain the respective undirected variants: the *undirected Planar*, the *undirected 2-Planar* and the *undirected Covington* transition systems. The transition set of each undirected transition system is as follows:

Undirected Planar

SHIFT	$\langle \sigma, i\|B, A\rangle \Rightarrow \langle \sigma\|i, B, A\rangle$
REDUCE	$\langle \sigma\|i, B, A\rangle \Rightarrow \langle \sigma, B, A\rangle$
ARC$_l$	$\langle \sigma\|i, j\|B, A\rangle \Rightarrow \langle \sigma\|i, j\|B, A \cup \{j \overset{l}{-} i\}\rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).

Undirected 2-Planar

SHIFT	$\langle \sigma_0, \sigma_1, i\|B, A\rangle \Rightarrow \langle \sigma_0\|i, \sigma_1\|i, B, A\rangle$
REDUCE	$\langle \sigma_0\|i, \sigma_1, B, A\rangle \Rightarrow \langle \sigma_0, \sigma_1, B, A\rangle$
ARC$_l$	$\langle \sigma_0\|i, \sigma_1, j\|B, A\rangle \Rightarrow \langle \sigma_0\|i, \sigma_1, j\|B, A \cup \{j \overset{l}{-} i\}\rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).
SWITCH	$\langle \sigma_0, \sigma_1, B, A\rangle \Rightarrow \langle \sigma_1, \sigma_0, B, A\rangle$

Undirected Covington

SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A\rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A\rangle$
NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A\rangle$
ARC$_l$	$\langle \lambda_1\|i, \lambda_2, j\|B, A\rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{j \overset{l}{-} i\}\rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).

We show in Figure 5 how the undirected Planar parser analyses a sentence using its own set of transitions. Note that the output of this parsing process is the undirected graph presented in Figure 3 instead of the expected dependency tree in Figure 2. In order to obtain a dependency tree as the final output of the analysis, we will need to apply a post-processing step, which will be described in Section 4.

It is worth remarking that, in order to apply this transformation to obtain an undirected parser from a directed one, the original transition system must satisfy the condition that their arc-building transitions (conventionally called LEFT-ARC and RIGHT-ARC) be identical except for the directions of the links that they create. This condition holds in some transition systems in the literature – such as the three systems described above, or the arc-eager DAG parser for enriched dependency representations described by Sagae and Tsujii (2008) –, but not in others. For example, in the well-known arc-eager parser by Nivre (2003), LEFT-ARC transitions pop a node from the stack in addition to creating an arc, while RIGHT-ARC transitions instead remove the topmost buffer node and then push the top stack node back

Transition	**Stack** (σ)	**Buffer** (B)	**Added Arc**
	[ROOT$_0$]	[This$_1$,...,tree$_5$]	
SHIFT	[ROOT$_0$, This$_1$]	[is$_2$,..., tree$_5$]	
ARC$_{SBJ}$	[ROOT$_0$, This$_1$]	[is$_2$,..., tree$_5$]	(1, SBJ, 2)
REDUCE	[ROOT$_0$]	[is$_2$,..., tree$_5$]	
ARC$_{ROOT}$	[ROOT$_0$]	[is$_2$,..., tree$_5$]	(0, ROOT, 2)
SHIFT	[ROOT$_0$, is$_2$]	[a$_3$,..., tree$_5$]	
SHIFT	[ROOT$_0$, is$_2$, a$_3$]	[dependency$_4$, tree$_5$]	
SHIFT	[ROOT$_0$, is$_2$, a$_3$, dependency$_4$]	[tree$_5$]	
ARC$_{NMOD}$	[ROOT$_0$, is$_2$, a$_3$, dependency$_4$]	[tree$_5$]	(4, NMOD, 5)
REDUCE	[ROOT$_0$, is$_2$, a$_3$]	[tree$_5$]	
ARC$_{DET}$	[ROOT$_0$, is$_2$, a$_3$]	[tree$_5$]	(3, DET, 5)
REDUCE	[ROOT$_0$, is$_2$]	[tree$_5$]	
ARC$_{PRED}$	[ROOT$_0$, is$_2$]	[tree$_5$]	(2, PRED, 5)
SHIFT	[ROOT$_0$, is$_2$, tree$_5$]	[ ]	
REDUCE	[ROOT$_0$, is$_2$]	[ ]	
REDUCE	[ROOT$_0$]	[ ]	

FIGURE 5. Transition sequence for parsing the sentence in Figure 1 using the Undirected Planar parser.

to the buffer. One could still try to transform the arc-eager parser into an undirected variant by converting each of its arc transitions into an undirected transition, without necessarily collapsing them into one, but this would result into a parser that violates the acyclicity constraint; because the original system is designed in such a way that both constraints are enforced jointly and acyclicity is only guaranteed if the single-head constraint is also kept. It is easy to check that this problem cannot happen in parsers where LEFT-ARC and RIGHT-ARC transitions are symmetrical in the manner described above: in these systems, if a given graph is not parsable in the original system, then its underlying undirected graph will not be parsable in the transformed system.

3.2. Undirected Feature Models

To implement a practical parser on top of a transition system, we need a feature model to extract relevant information from configurations that will serve to train a classifier. Therefore, apart from modifying the transition system, creating a practical undirected parser necessarily implies to adapt its feature model to work with undirected graphs.

Some features usually employed in transition-based parsers depend on the direction of the arcs that have already been created. Examples of such features are the part-of-speech tag associated with the head of the topmost stack node, or the label of the arc going from the first node in the buffer to its leftmost dependent.[1] However, since we cannot tell heads from dependents in an undirected graph, these features cannot be used to train an undirected parser.

Therefore, we convert these features into their closest undirected versions: in the previous examples, those would be the part-of-speech tag associated with the closer node linked to the topmost stack node, and the label of the arc that connects the first node in the buffer

[1] These example features are taken from the default model for the Planar parser in version 1.5 of MaltParser (Nivre et al., 2006).

to the leftmost node linked to it. Notice that now a node (topmost stack or first node in the buffer) has neither head nor dependents, it only has some other nodes linked to it.

More formally, these are the undirected features obtained from the directed ones:

- Information (e.g. part-of-speech, label, lemma, etc.) about the ith node linked to a given node (topmost stack node, topmost buffer node, etc.) on the left or on the right, and about the associated undirected arc, typically for $i = 1, 2, 3$,
- Information (e.g. part-of-speech, label, lemma, etc.) about the closest left and right "undirected siblings" of a given node, i.e., the closest node q located to the left of the given node p such that p and q are linked to some common node r located to the right of both, and vice versa. Note that this notion of undirected siblings does not necessarily correspond to siblings in the directed graph: it can also capture other second-order interactions, such as grandparents.

In addition, we create new features based on undirected relations between nodes that provide further context information for the parser. In particular, we found that the following features worked well in practice:

- A boolean feature representing whether two given nodes are linked or not in the undirected graph, and a feature representing the label of the arc between them.

4. RECOVERING ARC DIRECTIONS

The transformed transition systems described in Section 3 have the drawback that the output they produce is an undirected graph, like the one in Figure 3, rather than a proper dependency tree. In order to use these systems and still obtain a directed dependency tree as the final output of the parsing process, we will apply a post-processing step to assign an orientation to the undirected graph (i.e., choose a direction for each of its edges), in such a way that the single-head constraint is obeyed and the result is a valid dependency tree.

For this purpose, we have developed two different reconstruction techniques to recover arc directions from the undirected graph, previously described in less detail in Gómez-Rodríguez and Fernández-González (2012). The first one, called *naive reconstruction*, is based on using the dummy root node to decide the direction that should be assigned to edges, by choosing the unique orientation of the undirected graph obtained by traversing it from the dummy root. The second technique, *label-based reconstruction*, consists of using the edge labels generated by the transition system to assign a preferred direction to each undirected edge, and then choosing the orientation that conforms to as many preferred directions as possible (note that it will not always be possible to conform to the preferred directions of *all* the arcs, as that may generate a graph violating the single-head constraint).

To describe these reconstruction techniques more formally and view them under a common framework, we can formulate the problem of recovering arc directions as an optimum branching (i.e., directed minimum spanning tree) problem on a weighted graph. Given the undirected graph U produced by an undirected parser, we consider its isomorphic symmetric directed graph, i.e., the directed graph which has an arc for each of both possible directions of an undirected edge in U. Each directed spanning tree of that graph corresponds to an orientation of U. Then, reconstruction techniques can be implemented by assigning weights to each of the arcs in the symmetric graph, so that they encode a criterion to prefer certain orientations of arcs over others, and then using an optimum branching algorithm to find the minimum spanning tree. Different criteria for assigning weights to arcs will produce different reconstruction techniques.

More formally, let $U = (V_w, E)$ be the undirected graph produced by some undirected parser[2] for an input string w (we omit labels for simplicity and readability).

We define the following sets of arcs:

$$A_1(U) = \{(i,j) \mid j \neq 0 \wedge \{i,j\} \in E\},$$
$$A_2(U) = \{(0,i) \mid i \in V_w\}.$$

The set $A_1(U)$ contains the two possible orientations of each edge in U (i.e., the arcs in the symmetric directed graph isomorphic to U) except for those arcs that would have the node 0 as a dependent, which we disallow because we are using that node as a dummy root, and therefore it cannot be assigned a head. On the other hand, the set $A_2(U)$ contains all the possible arcs that link the nodes in V_w as dependents of the dummy root node, regardless of whether their underlying undirected edges were present in U or not. This is so that the reconstruction techniques defined under this framework are allowed to link unattached tokens to the dummy root.

With these arc sets, we define a graph $D(U)$ containing all the candidate arcs that we will consider when reconstructing a dependency structure from U:

$$D(U) = \{V_w, A(U) = A_1(U) \cup A_2(U)\}.$$

The reconstruction process for an undirected graph U consists on finding an optimum branching (i.e., a directed minimum spanning tree) for a weighted directed graph obtained from assigning a cost $c(i,j)$ to each arc (i,j) of the graph $D(U)$, that is, we are looking for a dependency tree $T = (V_w, A_T \subseteq A(U))$ that minimizes $\sum_{(i,j)\in A_t} c(i,j)$.

Such an tree can be calculated using well-known algorithms for the optimum branching problem, like the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). In this particular case, we can take advantage of the fact that the graph $D(U)$ has $O(n)$ nodes and $O(n)$ arcs for a string of length n, and use the implementation by Tarjan (1977) to achieve a time complexity of $O(n \log n)$.

The different reconstruction techniques can be defined by establishing different criteria to assign the costs $c(i,j)$ to the arcs in $A(U)$.

4.1. Naive Reconstruction

A first, very simple reconstruction technique can be defined by assigning costs to the arcs of $D(U)$ as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_1(U), \\ 2 & \text{if } (i,j) \in A_2(U) \wedge (i,j) \notin A_1(U). \end{cases}$$

This criterion assigns the same cost to both the orientations of each undirected edge in U, and a higher cost to attaching any node to the dummy root that was not directly linked to it in U. To obtain satisfactory results with this approach, we must train the undirected parser to explicitly build undirected arcs from the dummy root node to the root word(s) of each sentence using arc transitions. This means that, if our training treebank contains forests, we need to transform them into trees by explicitly linking each of their roots as dependents of the node 0, as explained at the end of Section 2.

Under this assumption, if no classification errors are made, the undirected graph U output by the undirected parser will always be an undirected tree, and the minimum spanning tree will correspond to the unique orientation of U making its edges point away from the

[2]Note that, while the approach taken in this article is to obtain undirected parsers by transforming directed parsers, it would also be possible in theory to design an undirected parser from scratch and apply the same reconstruction techniques to it.

dummy root.[3] It is easy to see that this orientation must be the correct parse, since any other orientation violates the assumption that node 0 is a root.

This naive reconstruction technique has the advantage of being very simple, while guaranteeing that the correct parse will be recovered if the undirected parser is able to correctly generate its underlying undirected tree. However, this approach lacks robustness, because it decides the direction of all the arcs in the final output based on which node(s) are chosen as sentence heads and linked to the dummy root. This means that a parsing error affecting the undirected edges that involve the root may propagate and result in many dependency links being erroneous. For this reason, this approach for recovering arc directions will not produce good empirical results, as will be seen in Section 5. Fortunately, we can define a more robust criterion where the orientation of arcs is defined in a more distributed manner, without being so sensible to the edges involving the root.

4.2. Label-based Reconstruction

To obtain a more robust and effective reconstruction technique, we first apply a simple transformation to the training corpus so that arcs will have their direction encoded as a part of their label. To do so, if a leftward arc in the training set is labelled X, we relabel it X_l, meaning "a leftward arc labelled X". If a rightward arc in the training set is labelled X, we relabel it X_r, meaning "a rightward arc labelled X".

After training the undirected parser with this modified treebank, its output for a new sentence will be an undirected graph where each edge's label includes an annotation indicating whether the reconstruction process should prefer to link the corresponding pair of nodes with a leftward or with a rightward arc. Note that those annotations represent *preferred directions* — not hard constraints — because, although in the absence of errors it would be possible to simply use the annotations to decode the correct parse for the sentence, in practice parsing errors can create situations where it is not possible to conform to all the annotations without violating the single-head constraint in the directed graph resulting from the reconstruction. In these cases, the reconstruction technique will have to decide which annotations to follow and which have to be ignored. For this purpose, we will assign the costs for our minimum branching algorithm so that it will return a tree agreeing with as many annotations as possible.

To achieve this, we denote by $A_{1+}(U) \subseteq A_1(U)$ the set of arcs in $A_1(U)$ that agree with the annotations, that is, arcs $(i,j) \in A_1(U)$ where either $i < j$ and $\{i,j\}$ is labelled X_r in U, or $i > j$ and $\{i,j\}$ is labelled X_l in U. Conversely, we call $A_{1-}(U)$ the set of arcs in $A_1(U)$ that disagree with the annotations, that is, $A_{1-}(U) = A_1(U) \setminus A_{1+}(U)$. Then, we assign costs to the directed arcs in $A(U)$ as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_{1+}(U), \\ 2 & \text{if } (i,j) \in A_{1-}(U), \\ 2n & \text{if } (i,j) \in A_2(U) \wedge (i,j) \notin A_1(U). \end{cases}$$

where n is the length of the string.

With these costs, the optimum branching algorithm will find a spanning tree that agrees with as many annotations as possible, since assigning the direction that agrees with an edge's annotation has a lower cost than assigning the opposite direction. Additional arcs from the root not appearing in the parsing output (i.e. arcs in $A_2(U) \setminus A_1(U)$) can be added, but only if this is strictly necessary to guarantee connectedness (i.e., if the graph U was disconnected),

[3] Note that, while we previously suggested using optimum branching algorithms to find the spanning tree for the sake of generality, in this particular case it is not necessary to use such a generic algorithm: the spanning tree can simply be built in $O(n)$ by starting a traversal from the root and orienting each arc in the sense of the traversal. However, this is only valid for this particular reconstruction technique.

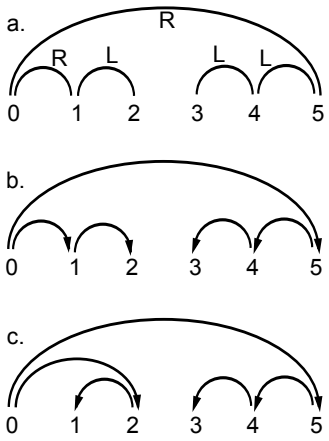

FIGURE 6. a) An undirected graph obtained by the undirected parser trained with a transformed corpus where arcs have been relabelled to specify their direction, b) and c) The dependency graph obtained by each of the variants of the label-based reconstruction (note how the second variant moves an arc from the root).

since the cost of such an arc ($2n$) is greater than the sum of costs of any combination of arcs originating from edges in U.

While this may be the simplest cost assignment to implement label-based reconstruction, we have found experimentally that better practical results are obtained if we give the algorithm more freedom to create new arcs from the root, as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_{1+}(U) \wedge (i,j) \notin A_2(U), \\ 2 & \text{if } (i,j) \in A_{1-}(U) \wedge (i,j) \notin A_2(U), \\ 2n & \text{if } (i,j) \in A_2(U). \end{cases}$$

The difference with the previous variant is that arcs originating from the root now have a cost of $2n$ *even if* their underlying undirected arcs were present in the output of the undirected parser. Informally, this means that the postprocessor will not "trust" the links from the dummy root created by the parser, and may choose to change them (at no extra cost) if this is convenient to obtain a better agreement with the label annotations of the rest of the arcs (see Figure 6 for an example of the difference between both cost assignments). We believe that the higher empirical accuracy obtained with this criterion is probably due to the fact that it is biased towards changing links from the root, which tend to be more problematic for transition-based parsers, while respecting the parser output as much as possible for links located deeper in the dependency structure, for which transition-based parsers have been shown to be more accurate (McDonald and Nivre, 2007).

Note that both variants of label-based reconstruction share the property that, if the undirected parser produces the correct labelled undirected graph for a given sentence, then the post-processing will transform it into the correct parse, which is simply the one obtained by following all the annotations in the undirected arcs.

4.3. Example

We can see how the reconstruction techniques work by going back to our running example sentence (Figure 2). In Section 2.3, we saw that the directed Planar parser could parse this sentence with the transition sequence shown in Figure 4. Then, at the beginning of Section 3, we illustrated how a wrong choice by the parser could cause error propagation: if an erroneous link from $dependency_4$ to $tree_5$ were created instead of the correct link from

$tree_5$ to $dependency_4$, the single-head constraint would then disallow creating the correct link from is_2 to $tree_5$, causing another attachment error.

If we parsed this sentence with the undirected Planar parser and the naive reconstruction, this error would never happen, since the undirected parser would simply not need to make the choice between linking $dependency_4 \rightarrow tree_5$ or $dependency_4 \leftarrow tree_5$. It would simply execute an ARC transition, as in Figure 5, and produce the undirected arc between $dependency_4$ and $tree_5$ that can be seen in Figure 3. The naive reconstruction technique would then extract from this graph the correct orientation (Figure 2), which is the one where every arc points away from the root.

On the other hand, if instead of the naive reconstruction we used the label-based reconstruction, the direction error in the example would translate into a labelling error in the undirected parser: instead of creating an undirected edge labelled $NMOD_L$ between the nodes $dependency_4$ and $tree_5$, the edge would be labelled $NMOD_R$, indicating a preference for right attachment. However, this preference would not be followed by the reconstruction technique, since there would be no possible way to conform to all the preferences at the same time without the node $tree_5$ getting two heads, and the only way of disobeying only one annotation (corresponding to the minimum spanning tree, with cost 15 in the second variant of label-based reconstruction) would be to disregard precisely that annotation and output, again, the parse of Figure 2.

Therefore, in this particular toy example, the combination of undirected parsing and any of the reconstruction techniques not only avoids the error propagation due to erroneously linking from $dependency_4$ to $\rightarrow tree_5$, but even eliminates the original error itself during post-processing. Of course, not all the cases will be so favorable when applying these techniques in practice, hence the need to evaluate them empirically to see whether undirected parsing can improve accuracy in real-life settings.

5. EXPERIMENTS

In this section, we evaluate the performance of the undirected Planar, 2-Planar and Covington parsers. For each transition system, we compare the accuracy of the undirected versions with naive and label-based reconstruction to that of the original, directed version. In addition, we provide a comparison to well-known state-of-the-art projective and non-projective parsers.

To evaluate the performance of the parsers in different languages, we use the following eight datasets from the CoNLL-X shared task: Arabic (Hajič et al., 2004), Chinese (Chen et al., 2003), Czech (Hajič et al., 2006), Danish (Kromann, 2003), German (Brants et al., 2002), Portuguese (Afonso et al., 2002), Swedish (Nilsson et al., 2005) and Turkish (Oflazer et al., 2003; Atalay et al., 2003); and we score the parsers on the following standard evaluation metrics:

- **Labelled Attachment Score (LAS):** The proportion of tokens (nodes) that are assigned both the correct head and the correct dependency relation label.
- **Unlabelled Attachment Score (UAS):** The proportion of tokens (nodes) that are assigned the correct head (regardless of the dependency relation label).

In our results, we show both LAS and UAS considering every token in the input sentences, including punctuation, as a scoring token.

In particular, Table 1 shows the results obtained by the undirected Planar parser with respect to the original Planar parser by Gómez-Rodríguez and Nivre (2010). Table 2 compares the results of the undirected 2-Planar parser to the 2-Planar parser by Gómez-Rodríguez

TABLE 1. Parsing accuracy of the undirected Planar parser with naive (UPlanarN) and label-based (UPlanarL) postprocessing in comparison to the directed Planar parser (Planar).

	Planar		UPlanarN		UPlanarL	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	67.34	77.22	66.33	76.75	**67.50**	**77.57**
Chinese	84.20	88.33	83.10	86.95	**84.50**	**88.35**
Czech	77.70	83.24	75.60	81.14	**77.93**	**83.41**
Danish	82.60	86.64	82.45	86.67	**83.83**	**88.17**
German	83.60	85.67	82.77	84.93	**85.67**	**87.69**
Portug.	83.82	86.88	83.82	87.06	**84.83**	**88.03**
Swedish	82.44	87.36	81.10	85.86	**82.66**	**87.45**
Turkish	71.27	78.57	68.31	75.17	**71.63**	**78.72**

TABLE 2. Parsing accuracy of the undirected 2-Planar parser with naive (U2PlanarN) and label-based (U2PlanarL) postprocessing in comparison to the directed 2-Planar parser (2Planar).

	2Planar		U2PlanarN		U2PlanarL	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	**67.19**	**77.11**	66.93	77.09	66.52	76.70
Chinese	**84.32**	**88.27**	82.98	86.81	83.94	87.75
Czech	77.91	83.32	75.19	80.80	**78.59**	**84.21**
Danish	83.61	87.63	81.63	85.80	**83.65**	**87.82**
German	85.76	87.86	82.53	84.81	**85.99**	**87.92**
Portug.	**84.92**	**88.14**	83.45	86.65	84.75	87.88
Swedish	**82.71**	**87.59**	80.71	85.68	82.25	87.29
Turkish	70.09	77.39	67.44	74.06	**70.64**	**77.46**

and Nivre (2010). Finally, Table 3 shows the results obtained by the undirected Covington non-projective parser in comparison to the directed implementation by Nivre (2008).

The results show that the use of undirected parsing with label-based reconstruction (UPlanarL) improves the scores of the Planar parser on all of the eight datasets tested. In most cases, it even attains higher scores than the 2-Planar baseline parser considered, which is remarkable if we take into account that the 2-Planar parser has more theoretical coverage due to its support of crossing links. In the case of 2-planar parsing, applying this technique (U2PlanarL) outperforms the scores of the directed 2-Planar parser on the Czech, Danish, German and Turkish datasets. Finally, the undirected Covington non-projective parser with label-based reconstruction (UCovingtonL) outperforms the results obtained by the baseline parser (Covington) on seven out of eight datasets.

The improvements achieved in LAS by undirected parsers with label-based reconstruction over the directed versions are statistically significant at the .05 level[4] for Danish, German and Portuguese for the Planar parser; and Czech, Danish and Turkish in the case of the Covington parser. Furthermore, no statistically significant *decrease* in accuracy was observed in any of the algorithm/dataset combinations.

[4]Statistical significance was assessed using Dan Bikel's randomized comparator: `http://www.cis.upenn.edu/~dbikel/software.html`

TABLE 3. Parsing accuracy of the undirected Covington non-projective parser with naive (UCovingtonN) and label-based (UCovingtonL) postprocessing in comparison to the directed algorithm (Covington).

	Covington		UCovingtonN		UCovingtonL	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	65.49	**75.69**	63.93	74.20	**65.81**	75.66
Chinese	85.61	89.62	84.02	87.73	**86.17**	**90.04**
Czech	77.43	83.15	74.78	79.92	**78.69**	**84.16**
Danish	82.89	87.06	81.61	85.51	**83.85**	**87.75**
German	85.69	87.78	83.51	85.39	**85.90**	**87.95**
Portug.	82.56	86.30	81.71	85.17	**82.70**	**86.31**
Swedish	**82.76**	**87.61**	81.47	85.96	82.73	87.23
Turkish	72.70	79.75	72.08	79.10	**73.38**	**80.40**

As expected, the undirected parsers with naive reconstruction (UPlanarN, U2PlanarN, UCovingtonN) performed worse than those with label-based reconstruction in all the experiments.

To further put these results into context, we provide a comparison of the novel undirected parsers, configured with label-based reconstruction, to well-known projective and non-projective parsers. Table 4 compares the undirected Planar parser to the arc-eager projective parser by Nivre (2003), a well-known algorithm that is also restricted to planar dependency structures.[5] The arc-eager parser is the default parsing algorithm in MaltParser (Nivre et al., 2006), and is also the dependency parser used in other current systems like ZPar (Zhang and Clark, 2011). In addition, Table 5 shows the results of the undirected 2-Planar and the undirected Covington algorithms compared to those of the arc-eager parser with the pseudo-projective transformation of Nivre and Nilsson (2005), which is able to handle non-planar dependencies, and is probably the most widely used method for non-projective transition-based parsing.

As we can see in these experiments, the undirected Planar parser obtains a better score than the arc-eager algorithm (MaltP) on seven out of eight tests; whilst the accuracy of the pseudo-projective arc-eager parser (MaltPP) is outperformed on five out of eight languages by the undirected Covington parser, and on the Arabic, Czech and Danish datasets by the undirected 2-Planar algorithm. Therefore, undirected parsing with label-based reconstruction can be used to improve the accuracy of parsing algorithms, and produces results that are competitive with state-of-the-art transition-based parsers.

For our tests, all the algorithms were implemented in MaltParser (Nivre et al., 2006), and trained with classifiers from the LIBSVM (Chang and Lin, 2001) and LIBLINEAR (Fan et al., 2008) packages. In particular, in order to reduce the training time for larger datasets, we employed the LIBLINEAR package for Chinese, Czech and German; and we used SVM classifiers from the LIBSVM package for the rest of the languages.

The arc-eager projective and pseudo-projective parsers were trained with the LIBSVM feature models presented in the CoNLL 2006 shared task, where the pseudo-projective version of MaltParser was one of the two top performing systems (Buchholz and Marsi, 2006). The feature models for the 2-Planar parser were taken from Gómez-Rodríguez and Nivre (2010) for the languages included in that paper. Regarding the new undirected parsers, their

[5]The arc-eager parser covers the set of *projective* dependency structures. Planar structures are a very mild relaxaction of projective structures, and in fact both sets become equivalent when sentences are required to have a single root node at position 0.

TABLE 4. Parsing accuracy of the undirected Planar (UPlanarL) with label-based postprocessing in comparison to the MaltParser arc-eager projective (MaltP) algorithm.

	UPlanarL		MaltP	
Language	LAS	UAS	LAS	UAS
Arabic	**67.50**	**77.57**	66.74	76.83
Chinese	84.50	88.35	**86.39**	**90.02**
Czech	**77.93**	**83.41**	77.57	83.19
Danish	**83.83**	**88.17**	82.64	86.91
German	**85.67**	**87.69**	85.48	87.58
Portug.	**84.83**	**88.03**	84.66	87.73
Swedish	**82.66**	87.45	82.44	**87.55**
Turkish	**71.63**	**78.72**	70.96	77.95

TABLE 5. Parsing accuracy of the undirected 2-Planar parser (U2PlanarL) and the undirected Covington non-projective parser (UCovingtonL) with label-based postprocessing in comparison to the MaltParser arc-eager pseudo-projective (MaltPP) algorithm.

	U2PlanarL		UCovingtonL		MaltPP	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	**66.52**	**76.70**	65.81	75.66	66.02	76.14
Chinese	83.94	87.75	86.17	**90.04**	**86.39**	90.02
Czech	78.59	**84.21**	**78.69**	84.16	78.47	83.89
Danish	83.65	**87.82**	**83.85**	87.75	83.54	87.70
German	85.99	87.92	85.90	87.95	**86.62**	**88.69**
Portug.	84.75	87.88	82.70	86.31	**84.90**	**87.95**
Swedish	82.25	87.29	**82.73**	87.23	82.67	**87.38**
Turkish	70.64	77.46	**73.38**	**80.40**	71.33	78.44

feature models for each algorithm and dataset were created from those of the directed parsers as described in Section 3.2.

6. ERROR ANALYSIS

The results in the previous section show that undirected parsing with label-based reconstruction can improve the accuracy of several parsing algorithms, and obtain results that are competitive with the state of the art in transition-based parsing. These results seem to support our hypothesis that the undirected parsing approach can successfully alleviate error propagation, formulated in Section 3.

To further test whether this is in fact the reason for the improvements in accuracy, we conduct a more in-depth analysis of the outputs produced by the parsers in Section 5, going beyond the global LAS and UAS metrics to see the circumstances under which parsing errors are being made.

In particular, as observed by McDonald and Nivre (2007), a good indicator of error propagation in transition-based parsers is the loss of dependency precision for longer dependency arcs (with the length of an arc $i \xrightarrow{l} j$ being defined as $|j-i|$). In most transition-based parsers, shorter arcs tend to be created earlier in transition sequences than longer ones: for example,

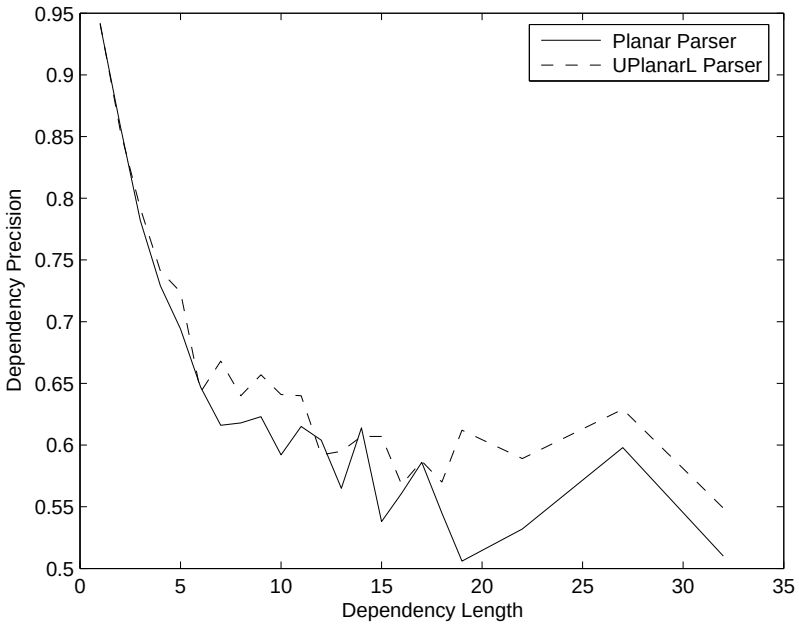

FIGURE 7. Dependency arc precision relative to predicted dependency length for the Planar parser (Planar) and the undirected Planar parser with label-based reconstruction (UPlanarL) on the eight datasets.

all the algorithms considered here (like the one used in McDonald and Nivre (2007)) have the property that given nodes $i < j < k < l$, an arc connecting j and k will always be created before an arc connecting i and l. This means that longer arcs, being created later, will be more likely to be affected by the propagation of errors made in previous parsing decisions.

Figure 7 shows the labelled dependency arc precision obtained by the directed and undirected planar parsers for different predicted dependency lengths, i.e., for each length l, it shows the percentage of correct arcs among the arcs of length l in the parser output. Figure 8 shows the labelled dependency arc recall for those same parsers as a function of gold dependency length, i.e., the percentage of gold standard arcs of each length l in the test set that were correctly predicted by the parser.

Instead of doing a language by language analysis, we measure these values across all datasets by aggregating the parser outputs for every language into a single file, and evaluating it with respect to a corresponding joined gold standard file, thus following the same method as McDonald and Nivre (2007). This ensures that we gain a better insight into the distribution of parsing errors, since the datasets for each individual language are too small to have a meaningful sample of arcs for each given distance. Note that, although this form of data aggregation means that the obtained precision and recall values are micro averages, the CoNLL-X test sets purposely have roughly the same size (5000 tokens), so the result would be roughly the same if we computed a macro average instead.

Starting from length 20, we group the data into bins of size 5 — $[20, 25)$, etc. — for a more meaningful visualization because, even with the mentioned aggregation, data become sparser from that point and there may be no arcs at all for some particular lengths.

As we can see in Figure 7, both the directed and the undirected planar parsers experiment a drop in precision for longer-distance dependencies, which is expected, among

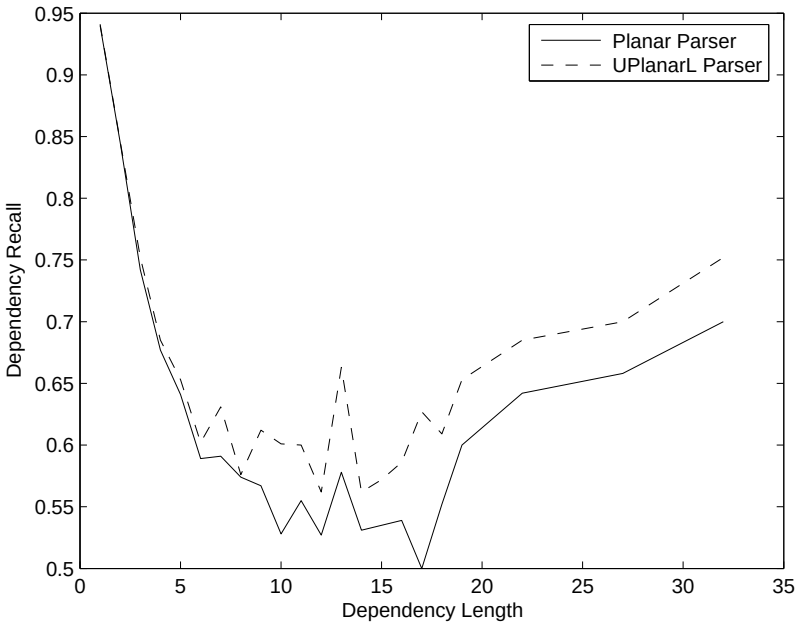

FIGURE 8. Dependency arc recall relative to gold dependency length for the Planar parser (Planar) and the undirected Planar parser with label-based reconstruction (UPlanarL) on the eight datasets.

other reasons, because both are transition-based parsers and will exhibit error propagation.[6] However, this phenomenon is significantly more pronounced in the directed parser, and while the accuracies of both algorithms are practically indistinguishable for arcs of length smaller than 5, the undirected Planar parser obtains a clearly better precision on longer arcs, with the difference in precision typically being above 3 points, and reaching up to 10 points on some lengths. Note that the difference in global LAS was not so huge (cf. Section 5) because shorter dependencies are more frequent in treebanks than longer ones, and thus they have a higher weight in the overall LAS metric.

The recall measurements, shown in Figure 8, exhibit a very similar trend. Note that, in principle, precision is a more useful metric than recall for the purpose of estimating the impact of error propagation, because the order in which dependency arcs are built is directly related to their relative length and position in the output trees, with the relation to the gold standard tree being more indirect. However, we include recall for completeness.

These results for the directed and undirected planar parsers suggest that, as we hypothesized, the undirected variant of the parser is less affected by error propagation than the original, directed version. This is the cause of the higher precision and recall for longer-distance

[6]While error propagation is probably the most important reason why a transition-based parser's performance drops for longer dependencies, it is not the only cause for this phenomenon: McDonald and Nivre (2007) observe that even in the graph-based parser by McDonald et al. (2006), which does not analyze sentences sequentially and thus cannot exhibit error propagation, there is a slight drop in accuracy for longer dependencies. A reason for this is that longer dependencies tend to occur more frequently in ambiguous constructions and in complex, relatively infrequent linguistic phenomena that are difficult to parse (Nivre et al., 2010), while short dependencies include many trivial instances like the attachment of determiners to nouns, where there is little ambiguity and many examples in training sets allowing for a high parsing accuracy.

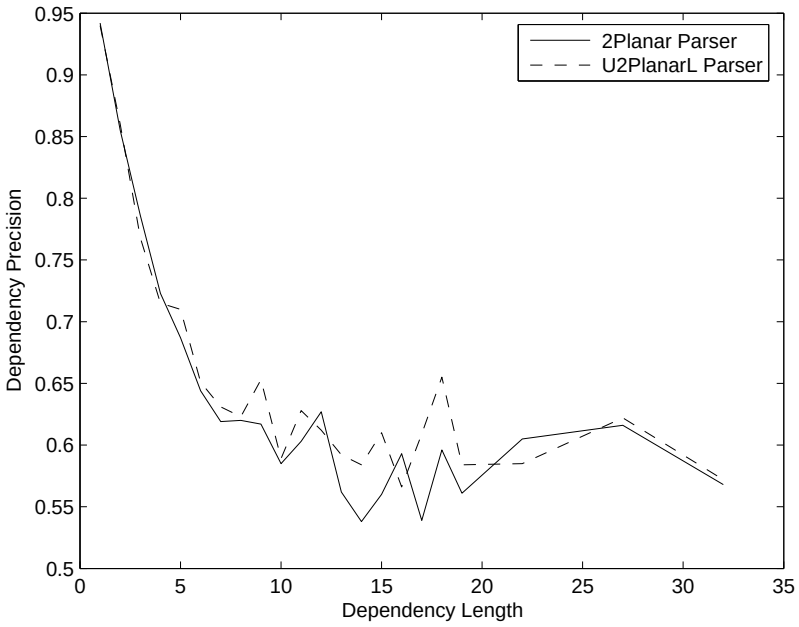

FIGURE 9. Dependency arc precision relative to predicted dependency length for the 2-Planar parser (2Planar) and the undirected 2-Planar parser with label-based reconstruction (U2PlanarL) on the eight datasets.

dependencies reflected in Figures 7 and 8, and of the improvement in overall accuracy that we observed in Section 5.

Figures 9 and 10 provide precision and recall measurements analogous to those in Figures 7 and 8, but this time for the directed and undirected 2-planar parsers. In this case, the differences between both parsers are not as marked as in the planar case, just as the diferences in overall accuracy were not as marked either (cf. Section 5). However, it is still clearly visible in Figure 9 that the undirected 2-Planar parser improves the precision for long-distance dependencies, supporting the hypothesis that the increases in accuracy that it produces are due to being less affected by error propagation.

Finally, Figures 11 and 12 provide the same comparison for the two variants of the Covington algorithm. In this case, the results are less conclusive, at least in the case of precision, with the undirected parser still clearly improving the recall for longer dependencies. This may indicate that the undirected Covington parser tends to generate more long arcs than the directed one (hence the improvement in recall but not in precision).

Summing up the results of the error analysis, we can see that undirected parsing clearly increases the precision for longer-distance dependency arcs, at least in the planar and 2-planar cases; providing evidence that this technique successfully alleviates error propagation. In the case of the Covington algorithm, the results are less conclusive, since there no clear increase (or decrease) in precision has been observed for longer dependencies.

7. CONCLUSION

In this article, we have presented a technique to transform transition-based dependency parsers satisfying certain conditions into undirected dependency parsers, which can then

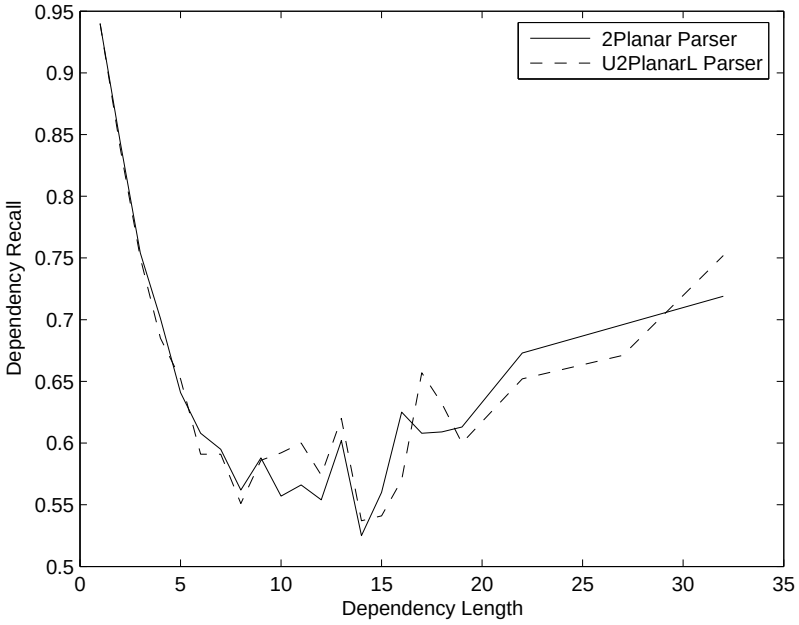

FIGURE 10. Dependency arc recall relative to gold dependency length for the 2-Planar parser (2Planar) and the undirected 2-Planar parser with label-based reconstruction (U2PlanarL) on the eight datasets.

be implemented and trained with feature models that do not depend on the directions of dependency links. The resulting parsers have the drawback that they generate undirected graphs instead of dependency trees, but we have shown how arc directions can be recovered from the undirected output by means of one of two different post-processing techniques, so that the final result is a fully functional dependency parser.

The advantage of the parsers obtained in this way is that they do not need to obey the single-head constraint until the post-processing step. This gives the parser more freedom when choosing transitions to apply, and alleviates error propagation, thus producing improvements in accuracy with respect to directed parsers. We have backed this claim with experiments in which we evaluated the directed and undirected version of the Planar, 2-Planar (Gómez-Rodríguez and Nivre, 2010; Gómez-Rodríguez and Nivre, 2013) and Covington (Nivre, 2008; Covington, 2001) parsing algorithms, obtaining improvements in labelled attachment score in 19 out of 24 algorithm-dataset combinations, with statistically significant differences for several of them, outperforming well-known state-of-the-art transition-based parsers. A more in-depth analysis has shown that the undirected parsers tend to perform especially well for longer-distance dependencies, which supports the hypothesis that the increase in accuracy is due to alleviation of error propagation.

The idea of parsing with undirected relations between words has been applied before in the work on Link Grammar (Sleator and Temperley, 1991). However, in that case undirected arcs are the desired final result, and not an intermediate result, since the Link Grammar formalism itself represents the syntactic structure of sentences by means of undirected links. To the best of our knowledge, the idea of obtaining an undirected graph as an intermediate step for parsing directed dependency structures has not been explored before in the literature.

Note that this article we have obtained undirected parsers by transforming existing di-

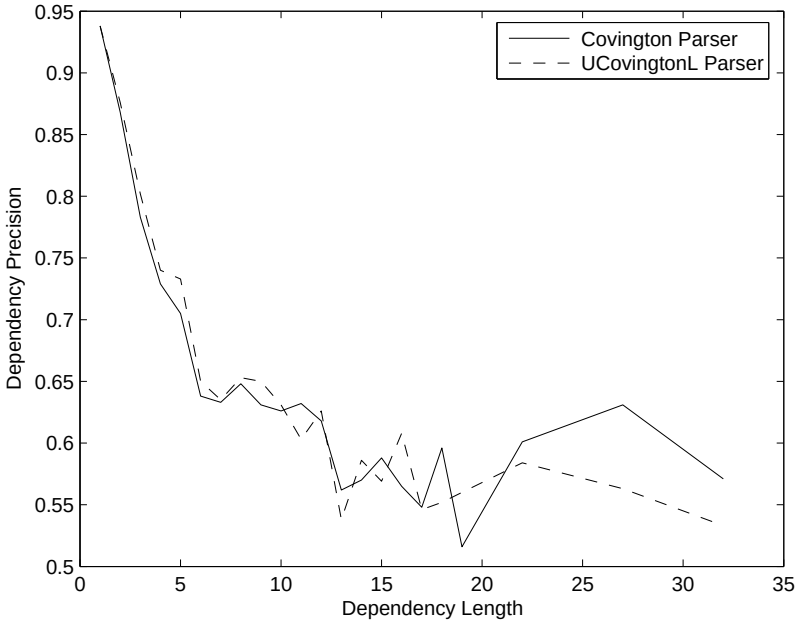

FIGURE 11. Dependency arc precision relative to predicted dependency length for the Covington parser (Covington) and the undirected Covington parser with label-based reconstruction (UCovingtonL) on the eight datasets.

rected parsers, and this provided a good baseline to assess the usefulness of the undirected parsing technique. However, it would also be possible to define undirected parsers from scratch, without necessarily being based on any directed parsers, and apply the same reconstruction techniques to them so as to obtain directed dependency structures as output. This is an interesting avenue for future work.

ACKNOWLEDGMENT

Partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (projects TIN2010-18552-C03-01 and TIN2010-18552-C03-02), Ministry of Education (FPU Grant Program) and Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Soc. do Coñec., Rede Galega de Proc. da Ling. e Recup. da Inf.). The experiments were conducted with the help of computing resources provided by the Supercomputing Center of Galicia (CESGA). We thank Joakim Nivre for helpful input in the early stages of this work.

REFERENCES

AFONSO, SUSANA, ECKHARD BICK, RENATO HABER, and DIANA SANTOS. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. *In* Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002), ELRA, Paris, France, pp. 1968–1703.

ATALAY, NART B., KEMAL OFLAZER, and BILGE SAY. 2003. The annotation process in the Turkish treebank. *In* Proceedings of EACL Workshop on Linguistically Interpreted Corpora (LINC-03), Association for Computational Linguistics, Morristown, NJ, USA, pp. 243–246.

ATTARDI, GIUSEPPE. 2006. Experiments with a multilanguage non-projective dependency parser. *In* Proceed-

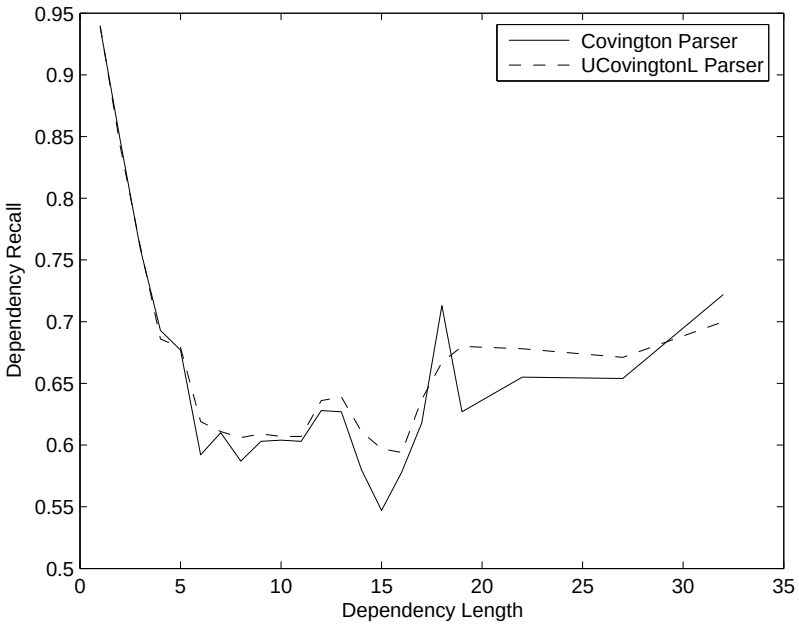

FIGURE 12. Dependency arc recall relative to gold dependency length for the Covington parser (Covington) and the undirected Covington parser with label-based reconstruction (UCovingtonL) on the eight datasets.

ings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 166–170.

BERANT, JONATHAN, IDO DAGAN, and JACOB GOLDBERGER. 2010. Global learning of focused entailment graphs. *In* Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, ACL '10, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 1220–1229. http://dl.acm.org/citation.cfm?id=1858681.1858805.

BRANAVAN, S. R. K., DAVID SILVER, and REGINA BARZILAY. 2012. Learning to win by reading manuals in a monte-carlo framework. J. Artif. Int. Res., **43**(1):661–704. ISSN 1076-9757. http://dl.acm.org/citation.cfm?id=2387915.2387932.

BRANTS, SABINE, STEFANIE DIPPER, SILVIA HANSEN, WOLFGANG LEZIUS, and GEORGE SMITH. 2002. The tiger treebank. *In* Proceedings of the Workshop on Treebanks and Linguistic Theories, September 20-21, Sozopol, Bulgaria. http://www.coli.uni-sb.de/~sabine/tigertreebank.pdf.

BUCHHOLZ, SABINE, and ERWIN MARSI. 2006. CoNLL-X shared task on multilingual dependency parsing. *In* Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 149–164.

CHANG, CHIH-CHUNG, and CHIH-JEN LIN. 2001. LIBSVM: A Library for Support Vector Machines. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

CHEN, K., C. LUO, M. CHANG, F. CHEN, C. CHEN, C. HUANG, and Z. GAO. 2003. Sinica treebank: Design criteria, representational issues and implementation. *In* Treebanks: Building and Using Parsed Corpora. *Edited by* A. Abeillé. Kluwer, pp. 231–248.

CHU, Y. J., and T. H. LIU. 1965. On the shortest arborescence of a directed graph. Science Sinica, **14**:1396–1400.

COMAS, PERE R., JORDI TURMO, and LLUÍS MÁRQUEZ. 2010. Using dependency parsing and machine learning for factoid question answering on spoken documents. *In* Proceedings of the 13th International Conference on Spoken Language Processing (INTERSPEECH 2010), Makuhari, Japan.

COVINGTON, MICHAEL A. 1990. A dependency parser for variable-word-order languages. Technical Report AI-1990-01, University of Georgia, Athens, GA.

COVINGTON, MICHAEL A. 2001. A fundamental algorithm for dependency parsing. *In* Proceedings of the 39th Annual ACM Southeast Conference, pp. 95–102.

CUI, HANG, RENXU SUN, KEYA LI, MIN-YEN KAN, and TAT-SENG CHUA. 2005. Question answering passage retrieval using dependency relations. *In* SIGIR '05: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA. ISBN 1-59593-034-5. pp. 400–407.

CULOTTA, ARON, and JEFFREY SORENSEN. 2004. Dependency tree kernels for relation extraction. *In* ACL '04: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA, pp. 423–429.

DING, YUAN, and MARTHA PALMER. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. *In* ACL '05: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA, pp. 541–548.

EDMONDS, JACK. 1967. Optimum branchings. Journal of Research of the National Bureau of Standards, **71B**:233–240.

EISNER, JASON M. 1996. Three new probabilistic models for dependency parsing: An exploration. *In* Proceedings of the 16th International Conference on Computational Linguistics (COLING), pp. 340–345.

FAN, R.-E., K.-W. CHANG, C.-J. HSIEH, X.-R. WANG, and C.-J. LIN. 2008. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research, **9**:1871–1874.

FUNDEL, KATRIN, ROBERT KÜFFNER, and RALF ZIMMER. 2006. RelEx—Relation extraction using dependency parse trees. Bioinformatics, **23**(3):365–371. ISSN 1367-4803.

GOLDBERG, YOAV, and MICHAEL ELHADAD. 2010. An efficient algorithm for easy-first non-directional dependency parsing. *In* Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT), pp. 742–750.

GÓMEZ-RODRÍGUEZ, CARLOS, and DANIEL FERNÁNDEZ-GONZÁLEZ. 2012. Dependency parsing with undirected graphs. *In* Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA. ISBN 978-1-937284-19-0. pp. 66–76.

GÓMEZ-RODRÍGUEZ, CARLOS, and JOAKIM NIVRE. 2010. A transition-based parser for 2-planar dependency structures. *In* Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, ACL '10, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 1492–1501. `http://portal.acm.org/citation.cfm?id=1858681.1858832`.

GÓMEZ-RODRÍGUEZ, CARLOS, and JOAKIM NIVRE. 2013. Divisible transition systems and multiplanar dependency parsing. Computational Linguistics, **39**(4):799–845.

HAJIČ, JAN, JARMILA PANEVOVÁ, EVA HAJIČOVÁ, JARMILA PANEVOVÁ, PETR SGALL, PETR PAJAS, JAN ŠTĚPÁNEK, JIŘÍ HAVELKA, and MARIE MIKULOVÁ. 2006. Prague Dependency Treebank 2.0. CDROM CAT: LDC2006T01, ISBN 1-58563-370-4. Linguistic Data Consortium.

HAJIČ, JAN, OTAKAR SMRŽ, PETR ZEMÁNEK, JAN ŠNAIDAUF, and EMANUEL BEŠKA. 2004. Prague Arabic Dependency Treebank: Development in data and tools. *In* Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools.

HAYASHI, KATSUHIKO, TARO WATANABE, MASAYUKI ASAHARA, and YUJI MATSUMOTO. 2012. Head-driven transition-based parsing with top-down prediction. *In* Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Jeju Island, Korea, pp. 657–665. `http://www.aclweb.org/anthology/P12-1069`.

HERRERA, JESÚS, ANSELMO PEÑAS, and FELISA VERDEJO. 2005. Textual entailment recognition based on dependency analysis and WordNet. *In* Machine Learning Challenges, *volume 3944 of* Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg-New York, pp. 231–239.

HUANG, LIANG, WENBIN JIANG, and QUN LIU. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. *In* Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1222–1231.

HUANG, LIANG, and KENJI SAGAE. 2010. Dynamic programming for linear-time incremental parsing. *In* Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 1077–1086.

JOHANSSON, RICHARD, and PIERRE NUGUES. 2006. Investigating multilingual dependency parsing. *In* Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 206–210.

JOSHI, MAHESH, and CAROLYN PENSTEIN-ROSÉ. 2009. Generalizing dependency features for opinion mining. *In* Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, ACLShort '09, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 313–316. http://dl.acm.org/citation.cfm?id=1667583.1667680.

KATRENKO, SOPHIA, PIETER ADRIAANS, and MAARTEN VAN SOMEREN. 2010. Using local alignments for relation recognition. J. Artif. Int. Res., **38**(1):1–48. ISSN 1076-9757. http://dl.acm.org/citation.cfm?id=1892211.1892212.

KATZ-BROWN, JASON, SLAV PETROV, RYAN MCDONALD, FRANZ OCH, DAVID TALBOT, HIROSHI ICHIKAWA, and MASAKAZU SENO. 2011. Training a parser for machine translation reordering. *In* Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP '11). http://petrovi.de/data/emnlp11b.pdf.

KROMANN, MATTHIAS T. 2003. The Danish dependency treebank and the underlying linguistic theory. *In* Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT), Växjö University Press, Växjö, Sweden, pp. 217–220.

LOMBARDO, VINCENZO, and LEONARDO LESMO. 1996. An Earley-type recognizer for dependency grammar. *In* Proceedings of the 16th International Conference on Computational Linguistics (COLING 96), ACL / Morgan Kaufmann, San Francisco, CA, USA, pp. 723–728.

MCDONALD, RYAN, KEVIN LERMAN, and FERNANDO PEREIRA. 2006. Multilingual dependency analysis with a two-stage discriminative parser. *In* Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 216–220.

MCDONALD, RYAN, and JOAKIM NIVRE. 2007. Characterizing the errors of data-driven dependency parsing models. *In* Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp. 122–131.

MCDONALD, RYAN, and JOAKIM NIVRE. 2011. Analyzing and integrating dependency parsers. Comput. Linguist., **37**:197–230. ISSN 0891-2017.

MCDONALD, RYAN, FERNANDO PEREIRA, KIRIL RIBAROV, and JAN HAJIČ. 2005. Non-projective dependency parsing using spanning tree algorithms. *In* Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), pp. 523–530.

MIYAO, Y., K. SAGAE, R. SÆTRE, T. MATSUZAKI, and J. TSUJII. 2009. Evaluating contributions of natural language parsers to protein-protein interaction extraction. Bioinformatics, **25**(3):394–400. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/25/3/394.

NILSSON, JENS, JOHAN HALL, and JOAKIM NIVRE. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. *In* Proceedings of the NODALIDA Special Session on Treebanks. *Edited by* P. J. Henrichsen.

NIVRE, JOAKIM. 2003. An efficient algorithm for projective dependency parsing. *In* Proceedings of the 8th International Workshop on Parsing Technologies (IWPT), pp. 149–160.

NIVRE, JOAKIM. 2008. Algorithms for Deterministic Incremental Dependency Parsing. Computational Linguistics, **34**(4):513–553. ISSN 0891-2017. http://www.mitpressjournals.org/doi/abs/10.1162/coli.07-056-R1-07-027.

NIVRE, JOAKIM, JOHAN HALL, and JENS NILSSON. 2004a. Memory-based dependency parsing. *In* Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004), Association for Computational Linguistics, Morristown, NJ, USA, pp. 49–56.

NIVRE, JOAKIM, JOHAN HALL, and JENS NILSSON. 2004b. Memory-based dependency parsing. *In* Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL), pp. 49–56.

NIVRE, JOAKIM, JOHAN HALL, and JENS NILSSON. 2006. Maltparser: A data-driven parser-generator for dependency parsing. *In* Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC), pp. 2216–2219.

NIVRE, JOAKIM, and JENS NILSSON. 2005. Pseudo-projective dependency parsing. *In* Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL), pp. 99–106.

NIVRE, JOAKIM, LAURA RIMELL, RYAN MCDONALD, and CARLOS GÓMEZ RODRÍGUEZ. 2010. Evaluation of dependency parsers on unbounded dependencies. *In* Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pp. 833–841.

OFLAZER, KEMAL, BILGE SAY, DILEK ZEYNEP HAKKANI-TÜR, and GÖKHAN TÜR. 2003. Building a Turkish treebank. *In* Treebanks: Building and Using Parsed Corpora. *Edited by* A. Abeillé. Kluwer, pp.

261–277.

SAGAE, KENJI, and JUN'ICHI TSUJII. 2008. Shift-reduce dependency DAG parsing. *In* COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics, Association for Computational Linguistics, Morristown, NJ, USA. ISBN 978-1-905593-44-6. pp. 753–760.

SHEN, LIBIN, JINXI XU, and RALPH WEISCHEDEL. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. *In* Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT), Association for Computational Linguistics, Morristown, NJ, USA, pp. 577–585.

SLEATOR, DANIEL, and DAVY TEMPERLEY. 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, Computer Science.

TAPANAINEN, PASI, and TIMO JÄRVINEN. 1997. A non-projective dependency parser. *In* Proceedings of the fifth conference on Applied natural language processing, ANLC '97, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 64–71. http://dx.doi.org/10.3115/974557.974568.

TARJAN, R. E. 1977. Finding optimum branchings. Networks, **7**:25–35.

TITOV, IVAN, and JAMES HENDERSON. 2007. A latent variable model for generative dependency parsing. *In* Proceedings of the 10th International Conference on Parsing Technologies (IWPT), pp. 144–155.

TRATZ, STEPHEN, and EDUARD HOVY. 2011. A fast, accurate, non-projective, semantically-enriched parser. *In* Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Edinburgh, Scotland, UK., pp. 1257–1268. http://www.aclweb.org/anthology/D11-1116.

XU, PENG, JAEHO KANG, MICHAEL RINGGAARD, and FRANZ OCH. 2009. Using a dependency parser to improve smt for subject-object-verb languages. *In* Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09, Association for Computational Linguistics, Stroudsburg, PA, USA. ISBN 978-1-932432-41-1. pp. 245–253. http://dl.acm.org/citation.cfm?id=1620754.1620790.

YAMADA, HIROYASU, and YUJI MATSUMOTO. 2003. Statistical dependency analysis with support vector machines. *In* Proceedings of the 8th International Workshop on Parsing Technologies (IWPT), pp. 195–206.

ZHANG, YUE, and STEPHEN CLARK. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. *In* Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 562–571.

ZHANG, YUE, and STEPHEN CLARK. 2011. Syntactic processing using the generalized perceptron and beam search. Computational Linguistics, **37**(1):105–151.

ZHANG, YUE, and JOAKIM NIVRE. 2011. Transition-based dependency parsing with rich non-local features. *In* Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2, HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA. ISBN 978-1-932432-88-6. pp. 188–193.